Industrial requirements for interactive product configurators

Queva, Matthieu Stéphane Benoit; Probst, Christian W.; Vikkelsøe, Per

Published in:
Proceedings of the IJCAI–09 Workshop on Configuration (ConfWS–09)

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Industrial requirements for interactive product configurators

Matthieu Quéva[*] and **Christian Probst**[*] and **Per Vikkelsøe**[†]
[*]DTU Informatics, Technical University of Denmark
{mq,probst}@imm.dtu.dk
[†]Microsoft Development Center Copenhagen
pvikkels@microsoft.com

Abstract

The demand for highly customized products at low cost is driving the industry towards Mass Customization. Interactive product configurators play an essential role in this new trend, and must be able to support more and more complex features. The purpose of this paper is, firstly, to identify requirements for modern interactive configurators. Existing modeling and solving technologies for configuration are then reviewed and their limitations discussed. Finally, a proposition for a future product configuration system is described.

1 Introduction

Increasing market demand concerning customization and market pressure from competitors force enterprises to adapt their production and selling processes. Indeed, today's customers demand products with lower prices, higher quality, and faster delivery, but they also want products customized to match their unique needs. In many industrial areas, Mass Production is nowadays replaced by Mass Customization [Pine, 1993], which provides customers with highly customized products and low unit costs. One of the essential tools enabling Mass Customization is a product configuration system. In a product configuration system, a configurable product is defined by a set of components, options, or more generally attributes that can be chosen by the user. Some of these attributes are bound together by constraints limiting the number of possible combinations. The configuration task thus takes as input a model representing the structure and the constraints of the product (*product knowledge*), and aims at finding a configuration satisfying all the constraints defined in the model, as well as the requirements given by the end-user. It can also output a price, or a specification of the product to be manufactured, usually as a *bill-of-materials* and *operations routes*. Interactive configurators display the possible combinations of the product's components and options to an end-user. When the user chooses among the possibilities, the configurator computes the consequences of these choices on the possible values available for the other attributes for example. In this paper, we will focus on this type of configurators. Two main challenges arise when dealing with interactive product configuration. At *modeling time*, there is a need to find efficient and easy-to-use ways for the design engineers to express the product knowledge. The more complex the product is, the more important this phase is. The second issue concerns how to solve, at *configuration time*, the constraints expressed at modeling time. When the different attributes of the product are instantiated, there is a need for an efficient solving engine, capable of solving all types of constraints defined previously. In such an interactive process, the end-user should be assisted through meaningful explanations and indications on how to satisfy his requirements in the case the solution is not directly available.
Solving the configuration problem has received a lot of attention from the research area [Amilhastre *et al.*, 2002; Mailharro, 1998; Mittal and Falkenhainer, 1990], while the modeling problem has been less covered [Aldanondo *et al.*, 2003; Felfernig *et al.*, 2002]. In this paper, we aim at identifying requirements for industrial use of product configurators. We then present a review of different techniques and technologies currently available for both modeling and solving the configuration problem. Finally, we propose new directions to explore for building state-of-the-art configurators.

2 Requirements analysis

In this chapter, we use a Home Multimedia Station (HMS) as a case study to discuss the requirements analysis for product configurators. The requirements are derived from various literature as well as discussion with industrial partners. We first present general requirements in the first section, followed by more specific features. For most of the requirements, an example is given through the case study.

2.1 General modeling requirements

A modeling environment for product configuration should:

- be *easy-to-use*: The persons that will interact with the modeling environment are usually design engineers, often possessing only basic programming skills. The modeling environment should therefore be accessible without advanced training in programming, and support easy development through tools for a fast implementation. Also, the terms used should be based on a widely accepted terminology, e.g. following Soininen et al.'s ontology of configuration [1998].

Markus Stumptner and Patrick Albert, Editors.
Proceedings of the IJCAI–09 Workshop on Configuration (ConfWS–09),
July 11–13, 2009, Pasadena, CA, USA.

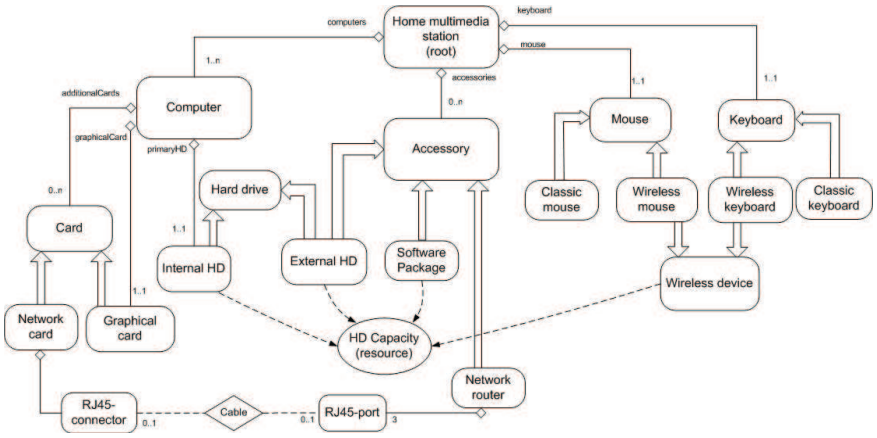

Figure 1: Structure of the Home Multimedia Station (HMS) case study[1]

- support *object-oriented modeling*: This approach has been favored by many researchers ([Hvam *et al.*, 2008; Rossi *et al.*, 2006]): it is indeed very suitable for product modeling, as product components can naturally be seen as objects. The object-oriented structure of the Home Multimedia Station can be seen in Figure 1.
- provide a *graphical representation*: This is important to the user for an easy understanding and a lower maintenance effort [Janitza *et al.*, 2003]. Graphical modeling languages will be presented in Section 3.
- be *extensible*: Companies use many applications around configurators. The system should provide an easy integration of CAD tools, databases, ERP or other systems in the configuration process.

2.2 Structure modeling

One part of modeling in configuration deals with representing the structure of the product. Several key features can be highlighted:

- *(dynamic) partonomy* (or part-of) relations define a subcomponent hierarchy in the product model. The multiplicity of these relations corresponds to the number of subcomponents to consider, which allows the reuse of component models. The possibility of specifying an unbound multiplicity permits to have a dynamic structure. The indefinite maximum in the "accessories" partonomy in the Home Multimedia Station models the possibility of adding a potentially unlimited number of accessories depending on the user's requirements.
- *taxonomy* (or kind-of/specialization) relations permit the use of generic base components to group features that are common to several subcomponents, which makes modeling and maintenance of the model much easier. Both the wireless and the classical mouses are specializations of the mouse component in the case study.
- *component groups* are a simple yet important feature in product modeling when it comes to product maintenance. Indeed, this makes it much easier to organize product knowledge data, as it allows to structure the model and its components according to specific criteria. The HMS model could be split into three groups: Computer Parts, Accessories, and Input Devices.
- definition of *units*: A product can be complex and can contain more than one data type with a specific unit. It should then be possible to declare different units, in order to make the model more realistic and the maintenance easier. The "price" of the Computer would be in *dollars*, while the "size" of the Internal HD would be in *inches*, although they are both real numbers.
- *connection ports* represent non-hierarchical relations between components that can be located in different subtrees of the model. Specific data can also be added to these relations, like the Cable component for the connection between the "RJ-45 Connector" and the "RJ-45 Port" in the HMS model.
- *default values* permit to provide the end-user with a capable default configuration very quickly, while still allowing the user to change some attributes.
- *hidden and locked attributes*: Using locked attributes to provide read-only information to the end-user or hidden attributes for internal computations offer increased flexibility to the model designer while reducing the complexity of the model for the customer.
- *production attributes*: Industrial product configurators are usually meant to be integrated with production management software, like ERPs. This includes mapping the configuration output to Bill-Of-Materials (BOMs) and operations routes that can be used in sales and manufacturing. Allowing the definition of production attributes that model how the BOMs and routes will be constructed from the product model's components is a great step towards an automatic generation of these production data.

[1] Part-of relations are represented with UML-like aggregations symbols and multiplicity, while taxonomy relations are represented using plain arrows. Dashed arrows represent use of resources.

Markus Stumptner and Patrick Albert, Editors.
Proceedings of the IJCAI–09 Workshop on Configuration (ConfWS–09),
July 11–13, 2009, Pasadena, CA, USA.

2.3 Constraint modeling

Another important aspect of product modeling is the definition of constraints on the model. Requirements for constraint modeling include:

- a panel of *built-in functions and constraints* available to the modeler: aside from simple arithmetic and logical constraints, advanced functions (e.g. sum), constraints (e.g. allEqual) or quantification (e.g. forAll) provide a much greater support to the product modeler.
- *table constraints*: More and more real product data is coming from tables representing allowed combinations of attributes/components. The ability to declare table constraints (or product catalogs) directly (instead of more complex formulas) simplifies by far the creation and usability of the model.
- *Continuous domains* for attributes give a much more precise representation of specifically tailored products for example, or just attributes involved in advanced arithmetic formulas.
- Products are often configured according to the *resources* they produce/consume. The modeling of these resources and how they increase/decrease must be defined to handle such cases. The HMS model involves a resource called "HD Capacity", which is produced by the hard drives and consumed by both wireless devices' drivers and software packages.
- *Soft (or prioritized) constraints* are constraints that may be violated if they are overridden by a user selection or indirectly as a consequence of a constraint with higher priority. Modeling with soft constraints permits to introduce a notion of uncertainty that can be used by the modeler to guide the configuration process with recommendation or simulate preferences for example. Such a constraint could be used in the HMS case study to recommend the user to choose a bigger internal hard drive if a specific software package is chosen.
- *layout constraints*: The same combination of components can result in different configurations when their layout is involved. One-dimensional positioning can be needed in the HMS model to order the list of cards in the computer, while more advanced positioning (2-D or 3-D) can be required to organize the disposition of the parts inside the computer box. Even more complex layout problems can be solved by an interaction with CAD tools [Aldanondo *et al.*, 2001].
- Defining *optimization (or cost) functions* helps the modeler specify how to calculate a value that then can be minimize or maximize at some point of the configuration, once the other user requirements are met. One cost function (to be minimized) declared in the HMS represents the overall price of the Home Multimedia Station as a function of the price of the components chosen.

2.4 Development and runtime support

Aside from product modeling, configurators also need to provide convenient tools to help understanding and solving the constraints defined in the model:

- The task of creating a product model is not only about defining the structure and constraints. Most of the modelers' time is usually spent in *debugging* the model, so that it behaves as it is intended to. Providing a convenient way to debug product configuration models is thus a priority for a product configurator.
- A must-have feature for a good configurator is the ability to provide *explanations* at configuration time. These explanations are given to the end-user when the configuration is over-constrained, or to provide guidance if he wants to force the selection of a value that is not allowed by the solving engine.

3 Product Knowledge Modeling

Product Knowledge Modeling represents a significant part in the configuration process. It consists in defining the model of a product family that will then be configured by the end-user. Development and maintenance of product knowledge bases are of primary importance, and the representation formalism must be thoroughly considered when choosing a product configuration system. Major vendors of configuration systems already use declarative knowledge modeling [Moller *et al.*, 2001].

Modeling languages are used to represent knowledge in a structured way. They can be categorized into two types: *graphical* and *textual* languages. Graphical languages use diagrams with symbols to express the different concepts, while textual languages use standardized keywords to structure the knowledge representation, that is then interpreted in an abstract syntax.

In the next sections, we will discuss two graphical languages, UML and SysML, both accompanied by a constraint textual language called OCL, as well as the textual modeling language EXPRESS.

3.1 UML and OCL

The *Unified Modeling Language (UML)* is an international standard defined in 1997 by the Object Management Group (OMG). This general-purpose object-oriented language is very well-known as it is widely used in industrial software development, and so is of prime choice for configuration.

The UML class diagram is worth our interest, as product modeling in configuration mainly deals with the structure and the constraints of the product. Several UML relations are of interests for configuration: the *association*, that establishes a semantical relationship between two components, and can be used to model *connection ports*; the *composition* (or composite aggregation), a parent-child relationship that can represent *partonomy* relations; and the *generalization*, used to model inheritance in UML for data and code reuse, and that can represent *taxonomy* relations in configuration.

Secondly, UML 2.0 contains an extension mechanism called *stereotypes*. A stereotype allows designers to extend UML by creating new model elements from existing ones. The new nodes are then stereotyped, which is reflected graphically by adding a name enclosed by guillemets above the name of another element. A stereotype can contains attributes, called tagged values.

Markus Stumptner and Patrick Albert, Editors.
Proceedings of the IJCAI–09 Workshop on Configuration (ConfWS–09),
July 11–13, 2009, Pasadena, CA, USA.

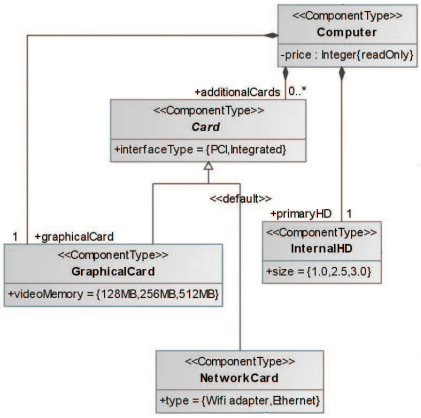

Figure 2: UML representation in the HMS case study

The *Object Constraint Language (OCL)* is an extension to UML that allows to write standardized constraints. It is actually a textual language that provides constraints and object query expressions that cannot be expressed using notations-like diagrams. OCL is a pure specification language, which means that an OCL expression will not have any side effects. Indeed, when an OCL expression is evaluated, it simply returns a value, and does not change anything in the model.

Product modeling using UML and OCL. Several researchers have showed interest in UML associated with product configuration. For example, UML is used by Hvam et al. [2008] in their "Procedure for building product models", along with another representation called *Product Variant Master*. They mainly focus on defining the object-oriented structure of the product model using UML, while more in-depth data (such as attributes and constraints) are stored in tables called *CRC cards*.
Felfernig et al. [2002] go further by defining a UML meta model architecture, i.e. a formalism to represent product configuration concepts and constraints using both UML and OCL. They automatically translate their model into an executable logical architecture, using the *XML Metadata Interexchange (XMI)* format - an XML-based OMG standard for the exchange of UML models.
The UML representation of a part of the Home Multimedia Station defined in Section 2 can be seen in Figure 2. This figure shows three partonomy and two taxonomy relations in the sub-model of the Computer. The OCL language can be used to describe constraints, e.g. that a computer that has a (primary) graphical card with 512 Mo of memory must cost at least 500 dollars:

```
context Computer inv:
self.graphicalCard.videoMemory = ''512Mo''
implies self.price >= 500
```

UML exhibits other interesting features for configuration, such as the use of packages. This allows the user to decompose its model into different groups of elements, thus permitting a better structure in the model.

UML/OCL limitations. The association UML/OCL provides an interesting object-oriented modeling experience, and the notoriety of UML among industry makes it an ideal candidate for product knowledge representation. As a graphical language, it also gives the design engineer a clear overview of the product model, making it more easy to see relationships between different components.
However, UML and OCL are not designed specifically for product modeling (and configuration), and thus miss interesting features. Although it is possible to adapt it to product configuration through the use of stereotypes, UML concepts are aimed at software engineering, and the transition can be difficult for the modeler. For example, it is not possible to declare units, and OCL falls a bit short when it comes to define table or soft constraints.
Finally, as a graphical language, its interpretation remains an issue. Indeed, the model must be interpreted in order to be integrated into a knowledge base and a configuration system. The work of [Felfernig *et al.*, 2002] goes in this direction, but more remains to be done in order to provide strong model checking and debugging facilities to the modeler.

3.2 SysML

The *Systems Modeling Language (SysML)* [SysML, 2001] is a recent modeling language developed as a joint initiative of OMG and the International Council on Systems Engineering (INCOSE). It is actually a UML profile, and thus inherits the characteristics of UML. The aim of SysML is to represent systems and product architectures, as well as their behavior and functionalities, where UML was used for software engineering. The development team of SysML aimed on the first hand at limiting the concepts too close from software engineering, and on the other hand at simplifying UML original notations by limiting the number of diagrams available, in order to make it easier to use.

Product Modeling using SysML. Modeling using SysML is very similar than doing so with UML, except that almost no user-defined stereotype is needed. Along with everything imported from UML, SysML defines *units* and *dimensions*. It is also possible to define objective (or optimization) functions and parametrized constraints using *Parametric diagrams* in SysML. This allows to represent constraints in diagrams where the parameters can be linked to the different attributes of the model's components, although the constraints' text still has to be expressed using OCL.
Finally, due to its full list of product-oriented diagrams, SysML gives the modeler the possibility to integrate the configuration model into a much wider product model, as UML does with software.

SysML limitations. Although SysML brings new capabilities relative to product modeling compared to UML, it still suffers from similar issues. For example, the constraints are still defined using OCL, and the matter of the interpretation of the model for model checking and debugging remains the same. The SysML extension provides more diagrams pre-stereotyped for product modeling (e.g. block, units, optimization functions,...), but still lacks some essential product con-

figuration concepts like resources for example.

3.3 STEP and EXPRESS

This section introduces the International Standard ISO 10303, which is referenced as *STEP (STandard for the Exchange of Product data)*. STEP was first released in 1994, and is published as a series of Parts. The goal of STEP is to allow the exchange of data describing a product between Computer Aided system (CAD, CAM, ...etc). It uses the EXPRESS language to formalize the semantics of the data, and the 20 series of Parts specify the standard data exchange mechanisms (e.g., data file or API access).
EXPRESS [EXPRESS, 2004] is thus an object-oriented data modeling language standardized as the Part 11 of STEP. It consists of two different representations: textual, or graphical (called *EXPRESS-G*). However, EXPRESS-G is not able to represent all details that can be formulated in the textual form, on which we will concentrate in this part.

Product Modeling using EXPRESS. Models in EXPRESS are organized according to *schemas*. These schemas permit to group the different elements of the model in relevant scopes, in the same way as UML packages. Components in EXPRESS are defined as *entities*, and are composed by *attributes*, that can be of basic types or entities themselves (partonomy). Taxonomy relations can also be represented through *abstract* classes and *subtypes*:

```
SCHEMA HomeMultimediaStationFactory;
  USE FROM ComputerParts;...
  ENTITY HomeMultimediaStation;
    price: DOLLAR;
    computers: SET[1:?] OF Computer; ...
  END_ENTITY;
  ENTITY Card ABSTRACT SUPERTYPE; ... END_ENTITY;
  ENTITY GraphicalCard SUBTYPE OF (Card); ... END_ENTITY;
  ...
END_SCHEMA;
```

Finally, named types and *units* can also be declared, clarifying the meaning and context of the variables of these types. Constraints can also be associated to each entities or types/units, through a *WHERE* clause.

```
TYPE DOLLAR = INTEGER;
WHERE SELF >= 0;
END_TYPE;
```

Although few built-in functions are available, EXPRESS allows *user-defined functions* using a full procedural programming language.

EXPRESS limitations. EXPRESS is a powerful language for product modeling, suitable for many product configuration problems. It contains nice features, such as units and constants declarations, as well as dynamic multiplicity or the possibility to define functions. However, the EXPRESS language is too general and is not suitable for knowledge engineers that are not expert in the language itself, mainly because of its lack of configuration-specific keywords. The definition of functions requires advanced programming skills, and writing a complex model without these functions can be very difficult or impossible, as a lot of functions are not built-in (min/max, sum, ...).

3.4 Discussion

Choosing a modeling language for product configuration is not a trivial task. Although graphical languages such as UML (and SysML) provide a clear and well-known representation of the product structure, interpreting the model is an issue, and advanced verification mechanisms may not be easy to built upon them. On the other hand, textual languages like EXPRESS provide a flexible formalism for modeling, but may be difficult to apprehend for a product modeler with few programming skills. Finally, specific features for product configuration are often missing, such as product catalogs and production attributes integration, or complex constraints (layout, soft, optimization functions, ...).

4 Solving the configuration

Proposing a language expressive enough to ideally model product families is not sufficient: the configuration system must be able to support this language and propose sufficient solving mechanisms. The combinatorial nature of configuration problems has led towards a wide use of *Constraint Satisfaction Problems (CSP)*.
Others topics of interest are model debugging and explanations. Indeed, both the modeler and the end-user must be assisted when using an interactive configurator. Modeler should be able to have a clear view of the running model and its constraints during design phase, while help should be provided to the end-user when he wants to force a value selection or when the configurator has reached an over-constrained choice with no solution.
In this section, we first recall the original definition of CSP and compare several dynamic extensions. We then review the trends in explanation generations and debugging.

4.1 Constraint Satisfaction Problems

The original CSP is a triple $\mathcal{P} = \langle X, D, C \rangle$ where:

- X is an n-tuple of variables $X = \langle x_1, x_2, ..., x_n \rangle$,
- D is a corresponding n-tuple of *domains* $D = \langle D_1, D_2, ..., D_n \rangle$, representing, for each variable x_i, the set of possible values it can take,
- C is a t-tuple of constraints $C = \langle C_1, C_2, ..., C_t \rangle$ restricting the values that the variables can simultaneously take.

Such problems are usually solved using search and consistency techniques. Search techniques are used to explore the solution space of the problem, the most famous one being *backtracking*, where the algorithm assigns each variable, tests for all the constraints, and then backtracks if no solution is to be found. Consistency techniques are used to reduce the domains of the variables during the solving, while keeping the problem consistent with the constraints. The most used consistency algorithm is arc-consitency (AC) 3, that research has been going on trying to improve (e.g. AC-3.3 in [Lecoutre *et al.*, 2003]). Other types of consistency techniques have also been investigated: path- [Bessière *et al.*, 2005] or k-consistency - although the complexity of the algorithms goes increasing. These techniques are aimed at problems on finite domains: working with continuous domains is

often more perilous [Benhamou *et al.*, 1999]. Finally, hybrid techniques can be used to improve the efficiency of consistency algorithms. Such techniques involve sharing of common subexpressions within the constraints [Hentenryck *et al.*, 1997], reshaping of the constraint network into a DAG (Direct Acyclic Graph), or reformulation of constraints [Benhamou and Granvilliers, 1997].

4.2 Extensions to CSP

Extensions to the classical CSP have been developed to permit the resolution of dynamic problems like product configuration. Mittal and Falkenhainer [1990] define a *Dynamic Constraint Satisfaction Problem (DCSP)*, where some variables are initially *active* while other not, and constraints are classified in two categories: *activity constraints*, that activate or deactivate variables, or *compatibility constraints* (similar to the constraints in the classic CSP).
This formulation includes algorithms only based on backtracking techniques, and has served as a basis for several others frameworks:

- a revision of the DCSP by [Soininen and Gelle, 1999], equivalent in complexity to the original CSP, and where activity constraints are generalized.
- Composite CSP (CCSP) [Sabin and Freuder, 1996] have been introduced to model a hierarchical structure between variables or constraints, using metavariables as placeholders for subproblems.
- In CSPe [Véron and Aldanondo, 2000], a state attribute is associated to each variable, giving the possibility to represent the activity of the variable, but can be also used for other purposes. The advantage of this formulation is that it can be solve using classic algorithms.
- More advanced algorithms for DCSP (renamed Cond-CSP) are developed in [Gelle and Faltings, 2003; Sabin *et al.*, 2003], and even further by Geller and Veksler [2005] with the ACSP.

Although well-studied, these dynamic CSP can only represent optional variables, and thus problems with an unbound number of variables cannot be solved with those methods. Two different approaches have thus been studied to solve that problem. Stumptner et al. [1998] describe the Generative CSP (GCSP), where constraints with metavariables can be used to express generic relations. Mailharro [1998] defines another framework, capable of satisfying on-demand generation of component in configuration. His approach is based on *constrained set variables*, which can contain a special value (*wildcard*) that represents the set of all components that have not been instantiated yet. Although a solving methodoly is presented, optimal algorithms in the number of value queries could be investigated.
One noticeable difference in these propositions is the representation of the product model using constraint satisfaction. Mailharro's approach focus on exploiting the component structure of the problem during solving. This kind of hybrid structure-based and constraint-based approach produces a constraint model much closer to the product model itself. This permits to design solving mechanisms specific to configuration, and that can reason on the structure of the product. It could also be of great help for giving debugging feedback to the modeler, thanks to its expressiveness.
The algorithms for solving these different problem representations are not always optimal, while time is a very important factor when dealing with interactive configuration. Precompilation techniques have thus been studied to circumvent those issues. The idea is to preprocess the constraint model at compile time into an efficient representation, using Binary Decision Diagrams [Hadzic *et al.*, 2004] or automata [Amilhastre *et al.*, 2002; Fargier and Vilarem, 2004]. However, generative problems or continuous domains are still an issue.

5 Explanations and Debugging

An efficient solving is not the only feature a constraint solver for interactive configuration should provide. Explanations and debugging support are necessary to help the user react intelligently when confronted to the modeling and configuration of a product.
Explanations are used to assist the end-user when answering the following questions: if there is a conflict, what are the reasons for inconsistency? Why is this feature selected by the solver engine? Why is this feature unavailable? Research work on explanations has recently increased, with the development of the QuickXplain algorithm by Junker [2004], that computes the minimal conflict set of a problem. In [2004], Friedrich proposes an improvement in the definition and the computation of the explanations in order to avoid the problem of *spurious explanations*. Recently, other approaches such as *corrective* [O'Callaghan *et al.*, 2005] and *representative explanations* [O'Sullivan *et al.*, 2007] have been developed to provide more intuitive explanations to the users.
Debugging support must also be provided to the user when it comes to test the product model. Indeed, an important part in the development of a model is actually verifying whether the model does what it is expected to do. Constraint debugging has been well studied in the research world, including visualization tools [Der, 2000] or the generic trace format from the *OADymPPaC project* [2007]. But few of these techniques have been specially targeted at product configuration, and are thus difficultly accessible to a classic design engineer. The concept of *model-based diagnosis* has been adapted to configuration by Felfernig et al. [2004] in order to tests the knowledge base with positive and negative test cases. Recently, Krebs [2008] proposed algorithms to identify relevant and irrelevant components for a specific product type (using segmentation of the product model tree), as well as detecting reachable component, using preprocessing to reduce the algorithm's complexity in some cases.
We strongly believe that research should focus on adapting debugging tools for product configuration, in order to alleviate the work of product modelers and testers, and limit the amount of time they spent on debugging their models.

6 Towards a future configuration system

In this section, we explore propositions for a new product configuration framework, based on the state-of-the-art technologies and the implementation of a new modeling language (Fig. 3).

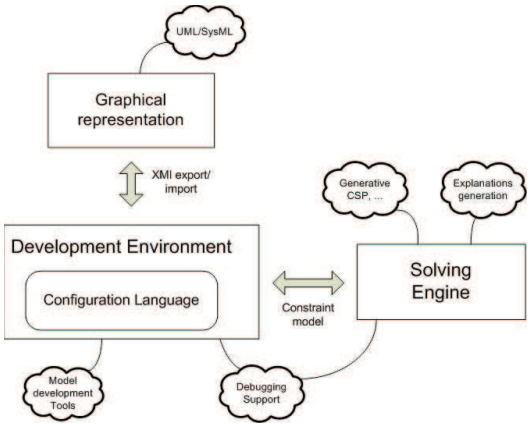

Figure 3: Product configuration framework

6.1 A new modeling language

As mentioned earlier, the language supporting the modeling of the product is of great importance for the configuration process. We propose the creation of a new declarative language that would be:

- *textual*: that type of language facilitates the interpretation and techniques like static analysis and optimization of the implemented model. Those techniques could for example be applied to perform model-checking, or propose correction of the model (e.g. by detecting unreachable values in attributes' domain). Integration with other systems such as a CAD software or a database would also be easier with such a language.
- *accessible* for model designers: the language should integrate keywords derived from configuration terminology, such as "product", "component", "constraint", "attribute" or even "BOM".
- providing a *graphical* representation: an export to a graphical language such as UML (or SysML) using the XMI format would permit to give a clear view of the model representation to the modeler, along with favoring the exchange of modeling data. Data about the model's constraints could be converted in OCL (when possible), or exported as text.
- designed according to the *modeling requirements* from Section 2, so that the language provides as many necessary features as possible for the model designer.
- aimed at enhancing designers' *productivity*: firstly, working with a textual language is often faster than through user interfaces, especially in an industrial environment where models are complex and contain a relatively big amount of data; secondly, such a language can be integrated in a development environment such as Microsoft Visual Studio, providing language services such as syntax-highlighting, code auto-completion, structured projects and many others. Last but not least, using a product modeling specific language would ease the integration with existing systems like ERPs or other tools (Word/Excel, ...).

6.2 Solving engine

The solving engine supporting the modeling language should at least implement the *Generative CSP* framework. Indeed, the dynamic possibilities of this framework are necessary to ensure the broadest panel of options available for the model. On the other hand, such a framework may not be as fast as simpler ones, and that is why other solutions may be envisaged. An interesting idea would be to perform an analysis of the model and determine what framework would be the most appropriate to use, improving execution times when possible. An explanation generation module will have to be integrated into the solving system. Such a module should integrate the recent state-of-the-art techniques, such as the *QuickXplain* algorithm [Junker, 2004] but could also experiment with corrective and representative explanations for example.

6.3 Advanced debugging support

Model-based diagnosis [Felfernig *et al.*, 2004] could provide interesting debugging options when designing a product model. The test cases could be gathered from previous configuration runs to make sure new model's additions do not create inconsistencies with old configurations. It is also worth investigating the automatic generation of the test cases: just after the creation of the model, by looking for potential weaknesses (e.g. targeting special values such as 0, infinity, or constructs like dynamic aggregations), or from a model proven correct, in order to test future modifications.
We also propose the exploration of debugging through *breakpoints*. The development of the model in an advanced environment such as Visual Studio gives the possibility to easily assign breakpoints to some parts of the model. Those breakpoints could target attributes and/or constraints, and be triggered when the attributes are modified, or the constraints provoke a change in the other assignments. A graphical overview of the constraint system could then be presented to the user.

7 Conclusion

Product configuration is a recent field of interest for both research and industry. As a consequence, the features and technologies needed for configuration systems are always evolving. We presented in this article a list of requirements for state-of-the-art product configuration systems, illustrated by a case study. We also described major existing modeling languages in the context of configuration, and discuss their limitations when it comes to configuration-specific features. Graphical languages fall short when it comes to automatic validation of the model. On the other hand, textual languages like EXPRESS are powerful but not suited for configuration model designers with few programming skills.
We then reviewed existing techniques for solving configuration problems, highlighting the need for advanced debugging support integrated with product modeling. As a solution, we made some propositions for a future product configuration framework, based on a new textual product modeling language integrated into a development environment. The development of this framework is the main objective of future work, associating static analysis, constraint solving and constraint debugging.

Markus Stumptner and Patrick Albert, Editors.
Proceedings of the IJCAI–09 Workshop on Configuration (ConfWS–09),
July 11–13, 2009, Pasadena, CA, USA.

References

[Aldanondo *et al.*, 2001] M. Aldanondo, J. Lamothe, and K. Hadj-Hamou. Configurator and cad modeler:gathering the best of 2 worlds. *IJCAI Configuration Workshop*, 2001.

[Aldanondo *et al.*, 2003] M. Aldanondo, K. Hadj-Hamou, G. Moynard, and J. Lamothe. Mass customization and configuration: Requirement analysis and constraint based modeling propositions. *Integr. Comput.-Aided Eng.*, 10(2):177–189, 2003.

[Amilhastre *et al.*, 2002] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic csps—-application to configuration. *AI*, 135(1-2):199–234, 2002.

[Benhamou and Granvilliers, 1997] F. Benhamou and L. Granvilliers. Automatic generation of numerical redundancies for non-linear constraint solving. *Reliable Computing*, 3(3):335–344, 1997.

[Benhamou *et al.*, 1999] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising hull and box consistency. *Proc. ICLP*, pages 230–244, 1999.

[Bessière *et al.*, 2005] C. Bessière, J.-C. Régin, R. HC Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *AI*, 165 (2):165–185, 2005.

[Der, 2000] *Analysis and Visualization Tools for Constraint Programming, Constraint Debugging (DiSCiPl)*, 2000.

[EXPRESS, 2004] EXPRESS. *10303-11 ISO - Part 11: The EXPRESS language reference manual*, 2004.

[Fargier and Vilarem, 2004] H. Fargier and M.-C. Vilarem. Compiling csps into tree-driven automata for interactive solving. *Constraints*, Vol. 9 Issue 4:263–287, 2004.

[Felfernig *et al.*, 2002] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Configuration knowledge representation using uml/ocl. *LNCS*, Jan 2002.

[Felfernig *et al.*, 2004] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *AI*, 152:213–234, 2004.

[Friedrich, 2004] G. Friedrich. Elimination of spurious explanations. *Proc. ECAI'04*, 2004.

[Gelle and Faltings, 2003] E. Gelle and B. Faltings. Solving mixed and conditional constraint satisfaction problems. *Constraints*, 8(2):107–141, 2003.

[Geller and Veksler, 2005] F. Geller and M. Veksler. Assumption-based pruning in conditional csp. *Proc. CP'05*, 3709:241–255, Oct 2005.

[Hadzic *et al.*, 2004] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. *Proc. PETO'04*, 2004.

[Hentenryck *et al.*, 1997] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: a Modeling Language for Global Optimization*. 1997.

[Hvam *et al.*, 2008] L. Hvam, N. H. Mortensen, and J. Riis. *Product customization*, volume XII. 2008.

[Janitza *et al.*, 2003] D. Janitza, M. Lacher, M. Maurer, U. Pulm, and H. Rudolf. A product model for mass-customisation products. *LNCS*, 2774:1023–1029, 2003.

[Junker, 2004] U. Junker. Quickxplain: preferred explanations and relaxations for over-constrained problems. *Proc. AAAI'04*, pages 167–172, 2004.

[Krebs, 2008] Thorsten Krebs. Debugging structure-based configuration models. *Proc. ECAI'08*, 2008.

[Lecoutre *et al.*, 2003] C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithm. *Proc. CP'03*, pages 480–494, 2003.

[Mailharro, 1998] D. Mailharro. A classification and constraint-based framework for configuration. *AI EDAM*, 12:383–395, Sep 1998.

[Mittal and Falkenhainer, 1990] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. 1990.

[Moller *et al.*, 2001] J. Moller, H. R. Andersen, and H. Hulgaard. Product configuration over the internet, 2001.

[OADymPPaC, 2007] OADymPPaC. *Generic Trace Format for Constraint Programming - Version 2.1*, Jan 2007.

[O'Callaghan *et al.*, 2005] B. O'Callaghan, B. O'Sullivan, and E. C. Freuder. Generating corrective explanations for interactive constraint satisfaction. *Proc. CP'05*, 2005.

[O'Sullivan *et al.*, 2007] B. O'Sullivan, A. Papadopoulos, B. Faltings, and P. Pu. Representative explanations for over-constrained problems. *Proc. CP'07*, 2007.

[Pine, 1993] B. J. Pine. *Mass Customization - The New Frontier in Business Competition*. Harvard Business School Press, 1993.

[Rossi *et al.*, 2006] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. 2006.

[Sabin and Freuder, 1996] D. Sabin and E. C. Freuder. Configuration as composite constraint satisfaction. pages 153–161, 1996.

[Sabin *et al.*, 2003] D. Sabin, E. C. Freuder, and R. J. Wallace. Greater efficiency for conditional constraint satisfaction. *LNCS - CP 2003*, pages 649–663, 2003.

[Soininen and Gelle, 1999] T. Soininen and E. Gelle. Dynamic constraint satisfaction in configuration. *Proc. of AAAI Workshop on Configuration*, Jan 1999.

[Soininen *et al.*, 1998] T Soininen, J Tiihonen, T Männistö, and R Sulonen. Towards a general ontology of configuration. *AI EDAM*, 12(4):357–372, 1998.

[Stumptner *et al.*, 1998] M. Stumptner, G. Friedrich, and A. Haselböck. Generative constraint-based configuration of large technical systems. *AI EDAM*, 12(4):307–320, 1998.

[SysML, 2001] SysML. *OMG Systems Modeling Language (OMG SysML) v1.0 Specification*, 2001.

[Véron and Aldanondo, 2000] M. Véron and M. Aldanondo. Yet another approach to ccsp for configuration problem. *Proc. ECAI'00*, pages 59–62, 2000.